CS 214: Systems Programming, Spring 2019
Extra Credit 0: Easter Egg Hunt

0. Abstract
This assignment is individual. Please feel free to dicuss it, but do not trade or offer code or exact information. For this assignment, you will duplicate the functionality of "ps aux". The command stands for process status and is a tool that displays a lot of useful information about running processes. Using the /proc entries, you can find all of the information ps prints out. It will however require a bit of digging and reading up on various data file formats in order to find all the little nuggets of information sprinkled about and hidden in the various 'files' under /proc.

1. Introduction
The utility was originally written by Branko Lankester as something of a hack. Since then it went through major revisions and about five authors, adding features like listing by PID, sorting, parsing from /proc and dirent integration. In order to build your own ps you should focus on the last two additions, namely using opendir() and readdir() to iterate through /proc and generating a line entry for each Process' information you find there.

As you may recall /proc is a 'directory' that has within it a directory for each Process' PID that is currently running. Each of those Process directories has in it a number of files that tell you about that Process' state. There are some elements that are repeated and some that are easier to parse out than others. The main difficulty is in figuring out where the information you need is contained in those 'directories' and then how to extract it. The man pages and the Internet are your friends in this regard.

2. Methodology and Implementation
You will need to open /proc, find each directory in it and open them in turn, opening data files you need and scraping out the information you need from them. One thing to keep in mind is the format of ps.

2.0 Right-justified
If you run it, you will notice all the text is right-justified along the column headings:

```
Left justified:        Right justified:
Heading                         Heading
data0                             data0
data1                             data1
longdatastuff             longdatastuff
```

That is however a fairly direct difference. You need to understand the various headings in ps.

2.1 PS columns
For the purposes of this project you will need to implment nearly all of the default ps headings:

```
[deymious@cp ~]$ ps aux | head -n 1
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

'USER' is the username of the owner of a Process. In actuality, this is one of the more difficult data values to determine. This may seem counterintuitive, however the machine does not care about a user's name. All users are given a user number, and that number is used internally as a user's identifier in the OS. It is only converted when displaying things to the user, however you need to know how to do that conversion, since you now need to interpret OS metadata for the user. In the spirit of Easter (?), I'll give you some pointers:

a. There is one file in the machine where username and user ID are stored and referred to; the password file. When a user logs in, they enter their username and the system needs to find their user number and verify them as that user.

b. You need to query the password file programatically. In other words, from within your code and not using the command prompt. As you can imagine, access to the password file is pretty well locked down and there is no way you can directly open() it. You will need to use a command to allow you look up something in it.

c. It sure would be nice if there was an entire library of **p**ass**w**or**d** commands that would allow you to **get** a **p**ass**w**ord file user struct, indexed by **uid**, and from there access the user's name.

'PID' is the Process ID of each Process in /proc. It should not be difficult to find.

'%CPU' is the amount of CPU time a Process has used divided by how long it has been running. You'll need to find out when the Process started and how much time passed between then and now. Keep in mind you don't have to format this as minutes and hours. It is the percentage of time the Process has used the CPU vs its runtime. It is also rounded to a tenths of a percent.

'%MEM' is the ratio of the Process' resident set size vs the physical memory present on the machine. It is a measure of the percentage of the machine's physical main memory the Process is currently taking up. For this you need to know how much virtual memory the Process' resident set is consuming; a **v**irtual **m**emory **r**esident **s**et **s**ize, so to speak, and the amount total in the machine. The machine's total physical memory is not something that changes per Process, so that is likely not going to be in an individual Process' entry. That kind of **mem**ory **info**rmation does exist elsewhere though under /proc.

'VSZ' is the virtual memory usage of the entire Process (in KB). This includes all the memory the Process is using, both resident and non-resident, or the Process' **v**irtual **m**emory **size**.

'RSS' is the Process resident set size, (in KB). You need this to figure out '%MEM', above.

'TTY' means “teletypewriter”, which was an early form of text-only fax. It was a way to encode electrical commands to a printer from a keyboard, so that you could type some text, encoding a document, and send those commands to any teletype to have it type out the document there. When you hear about the “AP newswire”, it used to be physically be a wire plugged in to a teletype. Since teletype was an existing standard for encoding keystrokes and typing actions (newline, tab, etc...) it was used as a way to encode user interaction for early computers. Naming terminal sessions after teletype ports stuck, so all textual command input is still referred to as 'tty' for historical reasons. Why the story? Don't worry about this one. The way security is done on the iLabs actually makes this fairly hard for you to do (if not impossible). You might notice that often even iLabs does not know, and tells you “?” for the TTY number sometimes. Just print out “?” for this one.

'STAT' is one or two letters/symbols that tell you the state of a process. It consists of one major code and one minor code. The major code is not tough, you can find it printed in a file for you. The minor codes are more problematic. You will need to deduce if they apply from the other things you can find in the /proc entries. Wouldn't be nice if you had the entire table of all the minors codes and their meanings? The manpages are your friends and they want to help. You will need to hunt fairly deep in the weeds to find the data that will decide all of them.

'START' is fairly direct. It tells you the time a Process was started, if it was started today, or the day/month it was started, if it was not started today. Be sure to output time in a 24-hour format and dates as a two digit day code folowed by a three character month abbreviation.

'TIME' is the amount of time this Process has run (*not* the time it has been alive/active/since start).

'COMMAND' is the command inputted in to the terminal and invoked that resulted in starting this Process. This can be somewhat annoying since actual executable name is not hard to find, but in order to find the arguments it was called with you need to dig around a bit.

3. Results

Submit an Asst2.5.tgz containing:
- a Makefile with at least the following directives:
    - 'all' to build your source in to a "my_ps" executable
    - 'clean' to delete all your compiled code
- a my_ps.c source files
- any other files or libraries you wrote to support your code

This assignment is individual. Please feel free to dicuss it, but do not trade or offer code or exact information.

It is not necessary to accept any command-line parameters. Presume your code will be writing output akin to ps called with the "aux" options at all times.

Image Credit: Envato Market, https://photodune.net/item/easter-basket/21613357

4. Appendix

Many Easter celebrations are based on a festivals of Eostre, a Germanic goddess of the dawn. The cycle of the year is renewed in spring when plants and animals return to growing and normal activity after the long winter months. Animals will breed and plants will bloom, which is codified and represented in the common association of chicks and rabbits as well as early spring flowers. Later Christian missionaries appropriated the holiday for ready extension to also symbolize the rebirth of Jesus. The Easter Egg Hunt is a continuance of the same, as the egg is taken as a metaphor for Jesus' tomb. Whether it had been part of the original fesitval is unclear, but it is widely accepted that Martin Luther, a monk and seminal figure of the Protestant Reformation, had held Easter Egg Hunts for women and children of the local parish as far back as the latter half of the 16th century. There is evidence of the idea of the eggs being hidden by a mythical 'Easter Bunny' around the 17th century. Over time much of the religious significance was lost and Easter became a largely secular holiday. The idea of an 'Easter Egg' became synonymous with finding something small, but fun or exciting, that either was hidden with the direct intention of being found, or was not necessarily intended to be found, but was put in an odd or difficult-to-access location.